Logique informatique et logique d'usage

Dinant, Jean-Marc

Published in:
Journal de Réflexion sur l'Informatique

Publication date:
1992

Document Version
le PDF de l'éditeur

Logique informatique et logique d'usage : former pour réconcilier

Introduction

Au seuil de la dernière décennie, une vague d'omni-informatisation a envahi la plupart des activités humaines de manière plus ou moins directe.

A la crête de cette vague l'ordinateur individuel a permis à des agents de formations diverses et non informatiques d'utiliser celui-ci dans leur travail quotidien. Traitement de textes, tableurs, grapheurs, Dbase et autres programmes du même type n'ont pu accéder à leur pleine prospérité que grâce à l'avènement du micro-ordinateur.

Au creux de cette vague, *la formation professionnelle des utilisateurs à l'informatique est longtemps resté le parent pauvre d'une évolution technologique époustouflante.* Si les concepteurs de matériel informatique ne se posent guère de questions sur l'évolution future des machines ordinateurs (de plus en plus rapides), «les concepteurs de logiciels ne savent pas ce qu'ils doivent fournir comme aide à l'apprentissage»[1].

Aujourd'hui, la confusion est grande entre informatique et ordinateur. En ce sens, durant les dernières décennies, les cours d'informatique commençaient par parler de code binaire, de cartes perforées, de bandes magnétiques, de claviers, etc... Une telle formation ne pouvait qu'être vouée à l'obsolescence avec une rapidité aussi fulgurante que celle de l'évolution du matériel.

A notre sens, il existe *plusieurs niveaux de formation* :

- un premier niveau de formation à la démarche informatique, que nous examinerons dans les trois premières parties de cet article, permet de définir un problème de dualité conceptuelle que nous détaillerons dans notre quatrième partie.
- un second niveau de formation, celui de la formation aux concepts d'une famille de logiciel, sera illustré en profondeur en prenant le concept de paragraphe dans la famille des traitements de texte.
- un troisième niveau - que nous n'aborderons pas ici - est celui de l'apprentissage d'un logiciel particulier.

Partant de l'informatique, nous verrons dans notre première partie l'indétermination latente du concept d'algorithme et par là-même plusieurs visions de ce que pourrait permettre l'informatique. Nous verrons comment, historiquement, s'est fait le glissement entre code et langage et comment ce glissement est générateur d'ambiguïté. Nous terminerons par un bref rappel des différentes phases de développement d'un projet informatique.

Partant de l'informaticien, nous examinerons ensuite en quoi consiste son travail, comment et pourquoi, il a créé des langages qui ne sont rien d'autre que *l'anthropomorphisation de transcodages.*

Partant de l'utilisateur, nous évoquerons alors dans notre troisième partie l'évolution considérable des interfaces au cours de ces dernières années.

Examinant en détail dans notre quatrième partie le problème de dualité contextuelle, nous verrons que plusieurs tendances dans l'évolution des interfaces tendent à réduire cette dualité contextuelle.

Dans notre cinquième partie, en prenant l'exemple d'une formation au traitement de texte, nous verrons qu'une formation conceptuelle, se basant sur la compréhension, non seulement du codage mais aussi des phases d'analyse, est une manière efficace et durable de réduire cette dualité contextuelle. Nous nous focaliserons sur la notion de paragraphe, bien moins triviale qu'il n'y paraît.

Enfin, nous conclurons par un certain nombre de recommandations permettant une formation efficace.

Activités humaines et machines algorithmiques

Lorsqu'un responsable de formation à l'informatique désire enseigner celle-ci, il commence bien souvent par introduire la distinction Software/ Hardware, continue en ne parlant que de code binaire, de bits, de bytes, de méga, de giga, de nano ou de micro... L'informatique est bien souvent présentée comme une machine à traiter l'information (Clavier -> processeur -> Ecran vidéo et imprimante)ou comme un processus de traitement de l'information (Données en entrées -> traitement -> résultat).

On constate le parallèle entre, d'une part une certaine perception de l'informatique, et, d'autre part, une machine qui rend possible une démarche intellectuelle. Bien souvent ce parallèle débouche sur une confusion. L'élève a en face de lui une machine et le seul but de la formation serait d'apprendre à s'en servir, c'est-à-dire à encoder les données en entrées et à lire les résultats.

Mais, comme le déclare Francis Pavé[2], *l'essentiel, c'est le logiciel,* c'est à dire le traitement pris en charge par le processeur comme décrit supra. Pour

faire comprendre ce traitement, le formateur introduira la notion d'algorithme : une suite d'opérations exécutables dans un ordre déterminé afin d'apporter une solution à un problème. Ces opérations sont des primitives c'est-à-dire des opérations directement exécutables par un automate auquel l'algorithme est destiné. Les qualités essentielles de cet automate est d'être *bête, discipliné et rapide*[3]. Trois caractéristiques que possèderait naturellement l'ordinateur.

C'est à ce moment précis que tout se joue et ce, d'une manière insidieuse, sur la notion d'algorithme et d'automate. Considérons deux exemples classiques d'algorithmes volontairement caricaturaux : l'un puisé dans un livre de cuisine, l'autre provenant des mathématiques.

Confection d'une omelette nature

1. Prendre deux oeufs
2. Casser les oeufs dans un plat
3. Battre les oeufs
4. Y rajouter une pincée de sel
5. Faire chauffer une poêle
6. Y faire fondre un peu de beurre
7. Y verser les oeufs battus
8. Laisser cuire à feu doux durant quelques minutes
9. Servir chaud

Dès lors, la vision que nous avons de l'algorithme (et donc quelque part de l'informatique) peut-être schématisée comme suit :

Oeufs
Sel ⇒ Automate + Algorithme ⇒ Omelette
Beurre

Les racines d'une équation du deuxième degré à une inconnue

Classiquement, on utilisera aussi comme illustration de l'algorithme la résolution des équations du deuxième degré. Problème que j'énonce brièvement dans un espace de nombre naturels.

Soient trois nombres A,B,C. Trouver la valeur de X tel que $AX^2 + BX + C = 0$
(les valeurs de X répondant à cette égalité sont appelées RACINES)

Les mathématiciens procèdent en deux étape.

- Calcul du Rho = $B^2 + 4AC$
- Application de la règle suivante :
  si Rho est supérieur à zéro : il y a deux racines
  si Rho est égal à zéro : il y a une racine (ou deux racines identiques)
  si Rho est inférieur à zéro : il n'y a pas de racines

Les résultats sont les suivants :

- Dans les deux premiers cas, les valeurs des racines sont données par :

$$X = \frac{-B \pm \sqrt{RHO}}{2A}$$

- Dans le troisième cas, il n'y a pas de racines.

Schéma de l'algorithme :

A
B ⇒ Automate +Algorithme ⇒ Racines de X
C

De la distinction entre code et langage

Dans les deux cas décrits ci-dessus, le rôle de l'opérateur, celui qui manipule la machine, semble simplifié. Nul besoin pour lui de concevoir des algorithmes complexes pour résoudre ses problèmes informationnels, quelqu'un d'autre s'en charge : l'informaticien, qui a conçu l'algorithme et l'a appris à l'ordinateur avec un langage qui ressemble de loin à une formule magique mais que nous démythifierons un peu plus loin. Cette démarche n'est pas loin des formules magiques ésotériques donnant vie au Golem[4] pétri de terre des temps anciens, même si le sable[5] a avantageusement remplacé la glaise. L'ordinateur ressemble alors à cette merveilleuse lampe d'Aladin capable d'exaucer nos moindres souhaits à condition de posséder les bonnes formules magiques ou, plus simplement, à condition de pouvoir les rédiger en terme d'algorithme.

Dans le premier exemple, il suffirait à l'utilisateur de communiquer beurre, oeufs et sel à la machine. Dans le deuxième il s'agit de lui donner des valeurs numériques de trois variables A, B et C. Dans les deux cas, le schéma conceptuel est clair et la formation à donner à l'utilisateur reste simpliste.

Et déjà, dans notre imagination[6], nous voyons un super robot ménager capable de préparer de succulentes omelettes. Et - pourquoi pas ? - capable d'appliquer toutes les recettes de cuisines... ou d'administrer une société... ou même de gouverner le monde[7].

Dans le deuxième exemple, nous voyons une machine étrange et puissante manipulant des formules hermétiques de plus en plus complexes. Bien au-delà des équations, elle peut calculer des trajectoires de satellites, simuler des expériences chimiques ou physiques à l'aide de modèles de plus en plus compliqués.

Et pourtant, si le calcul des racines d'une équation du second degré à une inconnue est un problème résolu depuis longtemps par informatique, nous ne connaissons pas énormément de machines à faire des omelettes, même si, ça et là, sont apparus dans l'univers de la ménagère divers systèmes (pas nécessairement informatiques, d'ailleurs), des outils qui ne prennent pas la place de la ménagère mais qui l'aident à divers stades de son travail (citons par exemple, la minuterie, le batteur électrique, etc.). L'omeletteur universel n'a encore aucune existence matérielle concrète. Serait-ce un problème de marché ? Difficile à croire. Un problème technique ? Etudions cette possibilité.

Saisir un oeuf (qui n'est qu'une des opérations nécessaires à la réalisation partielle du premier des neufs points composant la recette de cuisine) constitue une des tâches les plus difficiles à effectuer pour un automate. *Saisir un oeuf* n'est pas une primitive directement exécutable par un automate mécanisé assisté par ordinateur, pour deux raisons.

La première est liée à l'oeuf lui même : il est rond et fragile et donc très difficilement saisissable par un automate; cet obstacle n'est pas insurmontable d'un point de vue technique.

La deuxième raison est plus fondamentale et moins incontournable. Malicieusement, l'énoncé *casser deux oeufs dans un plat* regorge de sous-entendus et d'ambiguïtés que le lecteur aura rectifiés de lui-même. Il s'agit, *bien sûr,* d'un oeuf de poule et non d'autruche ou de pigeon, pas d'un oeuf en chocolat, ni d'un oeuf en plâtre, encore moins d'un oeuf de Colomb ou d'un neuf de pique. *Bien entendu,* l'oeuf est supposé être dans un état de fraîcheur raisonnable et tous ne seront pas d'accord pour définir d'une manière semblable la fraîcheur d'un oeuf. *Dans un plat* est une manière de parler. C'est le contenu des oeufs qu'il convient de verser dans le plat. Nul besoin de casser effectivement les oeufs dans le plat. Il est *bien évident* que la coquille ira dans la poubelle. Plus fort : si, en appliquant cette recette, vous voyez des écailles d'oeufs tombant dans le plat, vous allez *prendre l'initiative* de les retirer. Au nom de l'algorithmique ? Certes non! Au nom d'un *élémentaire bon sens* qui n'est pas contenu dans l'algorithme mais dans l'être humain qui l'exécute et qui, dès ce moment, cesse d'être un automate. Le bon sens et l'esprit d'initiative sont donc des éléments *parfois*[8] indispensables à la réussite de l'omelette. Deux éléments qui s'opposent au caractère bête et discipliné, au déterminisme de l'ordinateur.

Ainsi donc, si de nombreuses activités humaines sont détaillables sous forme d'algorithme, nombreuses sont celles dont l'algorithme n'est pas directement automatisable.

Ces deux visions caricaturales de ce que peut être un algorithme nous permettent de porter notre attention sur un concept fondamental qui sous-tendra la suite de l'exposé : celui de langage et, sous-jacent, celui de communication. Il y a là un langage double : celui que l'informaticien devra utiliser pour communiquer un programme à la machine et celui que l'utilisateur devra utiliser pour se faire comprendre de celle-ci lorsqu'elle sera gouvernée par le programme de l'informaticien. La formation des utilisateurs serait donc l'apprentissage d'un nouveau type de langage, mais cet apprentissage ne peut pas se limiter à un dictionnaire de commandes. Avant même d'aborder l'inévitable opérationalité (c-à-d le code des commandes à exécuter pour effectuer certaines opérations), l'utilisateur doit avoir parfaitement intégré que les règles qui régissent la communication entre une machine et lui-même sont différentes de celles, qui lui sont familières, d'un dialogue entre êtres humains. Il s'agirait donc moins d'apprendre des mots que d'*apprendre comment parler à un informaticien et comment l'écouter par machine interposée.*

Dans ce qui suit, nous nous proposons de montrer que l'utilisation d'un programme par un utilisateur est un *processus de communication réduit et altéré* entre un informaticien et un utilisateur. Un peu comme si ceux-ci, se trouvant à deux étages différents, s'échangeaient des messages stéréotypés à grands renforts de frappe de clavier, de coups de souris et de crépitements d'imprimante. Conceptuellement, un ordinateur n'est rien d'autre qu'un énorme répondeur téléphonique qui va sélectionner sa réponse selon le message de l'utilisateur.

Pour cela, nous examinerons tout d'abord, dans les grandes lignes de la genèse d'un logiciel, la démarche qui préside au comportement de l'informaticien.

Les phases de développement d'un programme

L'analyse de l'existant

Abandonnons l'idée mythique de l'informaticien en bras de chemise pianotant des formules magiques sur son clavier jusque très tard dans la nuit. La première démarche de l'informaticien est de construire un *modèle* de la réalité à informatiser. Ce modèle prend corps dans une description formelle des informations pertinentes à traiter (les données) et des différents traitements à effectuer en vue de produire des résultats.

Prenons l'exemple simple de la gestion d'un fichier d'adresses. L'informaticien, après analyse, «décidera», par exemple, qu'une adresse se compose d'un nom, d'une rue, d'un numéro, d'un code postal et d'un nom de ville. Ensuite, il décidera du format physique de l'adresse. Un nom pourra avoir 50 caractères au maximum, le code postal sera composé de 4 chiffres, etc. Ce choix n'est le fruit de l'arbitraire de l'informaticien mais le résultat d'une analyse du fichier d'adresses existant. Les traitements à effectuer seront relativement simples : l'utilisateur devra, par exemple, pouvoir supprimer, ajouter ou modifier une adresse.

Cette liste est fatalement *exhaustive*. Le programme résultant de cette démarche ne pourra, en effet, qu'exécuter ces *primitives*. Dans cette étape, l'informaticien interrogera de futurs utilisateurs (par exemple des dactylos pour un traitement de texte).

L'analyse fonctionnelle

Dans un deuxième temps, d'abord mentalement, par écrit ensuite, il va élaborer différents algorithmes qui lui permettront de mettre à la disposition de l'utilisateur futur un ensemble de fonctions appelables par des modes opératoires. Une partie de sa réflexion portera sur l'interface utilisateur qu'il devra mettre au point. Quelles sont les séquences de touche que l'utilisateur devra pousser afin d'exécuter telle ou telle action ? L'informaticien se fait une certaine idée de l'utilisateur futur et de la manière dont il va réagir. Il peut, suivant l'idée qu'il s'en fait, l'obliger à une plus ou moins grande prudence. Ainsi, par exemple, lors de la suppression d'une adresse, il pourra demander à l'utilisateur de confirmer sa volonté d'effacer une adresse du fichier d'adresses en demandant à l'ordinateur d'imprimer à l'écran «Etes-vous bien certain de vouloir effacer cette adresse (O/N)? ». C'est lui qui décidera que faire si l'utilisateur ne tape ni sur la lettre "O" (pour dire oui) ni sur "N" (pour dire non). Il peut supposer que l'utilisateur n'a pas compris la question et demander à l'ordinateur d'imprimer dans ce cas une phrase du type «Répondez "O" pour oui ou "N" pour non». Il peut aussi faire en sorte que l'ordinateur émette un «beep» sonore afin d'attirer l'attention de l'utilisateur. Il lui est loisible de faire en sorte que l'ordinateur prenne toute touche qui n'est pas un "O" pour un non, etc... Cette phase s'appelle l'*analyse fonctionnelle*. Le travail d'analyse de l'informaticien comprend donc deux volets : il s'agit d'abord d'observer et de comprendre la réalité observée puis, dans un deuxième temps, de déterminer mentalement des procédés informatiques capables d'automatiser les flux d'informations identifiés dans le premier volet.

Soulignons deux éléments de cette attitude de l'informaticien-analyste.

Tout d'abord, le comportement de l'informaticien n'est pas seulement une démarche analytique mais aussi une démarche spéculative dans la mesure où il doit, à l'avance, imaginer et interpréter l'attitude de l'utilisateur. Il établit certaines conventions que l'utilisateur devra respecter s'il veut être «compris» de la machine. C'est le programmeur qui fixe «les règles du jeu», le *code* de la communication. Le dialogue se trouve altéré si l'utilisateur ne respecte pas les conventions qui lui sont données. Or un objectif majeur de la formation est précisément d'apprendre à l'utilisateur un code qui ne correspond pas à son intuition ou à son expérience ni aux codes qu'il a l'habitude d'utiliser pour se faire comprendre d'une machine dans d'autres contextes. La richesse de la communication qui s'établira entre la machine et l'utilisateur s'en trouve réduite. En ce sens, *le processus de communication est altéré*.

Deuxièmement, le programme construit n'est pas à même d'interpréter de façon correcte toute pression de touche sur le clavier. Il considérera uniquement les touches qui font partie des situations que le programmeur a prévu. L'ordinateur, par sa nature de machine, ne recevra comme message que les signaux électriques produits par une pression de touche du clavier. Une pression nerveuse produit exactement le même effet qu'une pression calme ou hésitante. Cela signifie que la machine est incapable de prendre

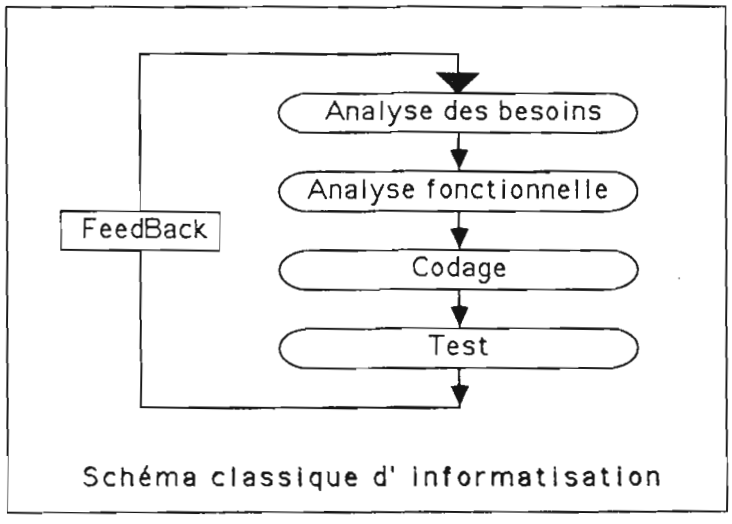


Schéma classique d' informatisation

en considération le contexte dans lequel va se dérouler le programme. En ce sens, *le processus de communication est limité.*

Le codage

Le codage constitue la troisième étape du travail de l'informaticien, la partie visible de l'iceberg de la programmation prise au sens large. Il a pour but de communiquer à la machine, par l'intermédiaire d'un code appelé langage, des normes d'un comportement futur. La machine est programmée afin de réagir d'une manière déterministe à certaines actions de l'utilisateur qui seront converties par la machine en bit et en byte. Cette phase est souvent celle qui représente dans l'esprit du grand public le travail de l'informaticien et le mot magique *programmer* jouit d'une auréole sans autre pareille. C'est la phase la plus spectaculaire car c'est à ce moment que le programme prend forme et devient capable de réagir mais la qualité du logiciel dépendra principalement de la qualité des deux étapes en amont. C'est le moment précis ou l'informaticien anime la machine et lui donne une autonomie propre, le stade où il inscrit les paroles magiques sur le front du golem.

Les tests et la maintenance

Vient alors une phase de test qui peut être l'occasion d'un rééquilibrage des rôles joués par l'utilisateur et l'informaticien. Bien souvent, dans le cas de logiciels créés sur mesure, l'informaticien interrogera l'utilisateur lors de la première phase d'étude de l'existant, non seulement pour comprendre sa manière de procéder mais aussi pour tenir compte de ses desiderata pour l'architecture future du système informatisé. La phase d'analyse fonctionnelle nécessite des compétences qui sont dans la grande majorité des cas hors de sa portée.

Ensuite, durant toute son existence, l'utilisation d'un logiciel mettra au grand jour un certain nombre de lacunes. Il peut s'agir d'erreurs issues d'une des trois étapes que nous venons de décrire. Normalement, les erreurs les plus graves auront été détectées et corrigées lors de la phase de test. Les erreurs les plus connues sont les «bogues» (on parle alors de «debugging» ou «deboggage»); il s'agit d'erreurs de programmation qui ont souvent comme effet de provoquer l'arrêt instantané du programme. Les erreurs de programmation sont peu à peu identifiées et réparées. Contrairement à l'idée largement répandue (par les informaticiens ?), les pannes techniques, liées au fonctionnement électronique de l'ordinateur, sont relativement rares. Il s'agit le plus souvent de pannes logicielles dues à une erreur humaine de l'informaticien en amont. De quelque type qu'elle soient, ces erreurs viennent généralement du fait que l'utilisateur accomplit une action ou, plus souvent, une combinaison d'actions que l'informaticien n'avait pas prévue. Ainsi, on peut citer l'histoire de cet homme de cent et six ans qui a reçu son certificat d'inscription à l'école primaire. Ce fait divers authentique est dû au fait que l'informaticien avait prévu d'utiliser deux chiffres pour inscrire la date de naissance dans le fichier de la population. Il avait oublié de prendre en compte le cas de personnes dont l'âge dépasse la centaine. Lors de l'impression des formulaires d'inscription, le programme avait simplement sélectionné les personnes dont la date de naissance était antérieure de six ans à l'année actuelle, ne faisant aucune différence entre 1982 et 1882 puisque l'année de naissance se résume dans ces deux cas à 82. Des erreurs de ce genre sont quasiment inévitables, ce qui explique la nécessité de la *maintenance* des programmes.

Du côté de l'informaticien : du mnémonique au langage de programmation

Mémoires et codification binaire

Au début, il n'y avait pas d'ordinateurs mais des calculateurs. Ceux-ci ressemblaient à la pascaline, au détail près que la mécanique est remplacée par du courant électrique. Pour additionner deux nombres, il convenait d'établir des connections électriques entre plusieurs circuits. Cela se faisait via un opérateur humain spécialisé. La machine s'est considérablement améliorée à partir du moment où on a pensé à stocker dans une *mémoire* les nombres à additionner. Ensuite, le nombre d'opérations envisageables augmente : on peut non seulement additionner mais aussi soustraire, multiplier et diviser. Dès lors, il serait intéressant d'utiliser la mémoire non seulement pour stocker les données (les nombres sur lesquels on désire effectuer des opérations) mais aussi les instructions (additionner, soustraire, multiplier, diviser). On découvre à ce moment le concept de programme. Techniquement, il reste à inventer la mémoire de masse pour pouvoir stocker définitivement données et programmes et - pourquoi pas ? - les résultats qui pourront ainsi être à leur tour données d'autres programmes. Il reste aussi à améliorer les organes d'entrée permettant de communiquer la valeur des données et la liste des opérations à établir et les organes de sorties (en entrée : cartes perforées, ensuite clavier et plus tard (±1986) la souris; en entrée et en sortie l'écran vidéo; en sortie carte perforées et imprimante,...).

Dès l'origine, les informaticiens ont du imaginer un *code* binaire. Comment faire avec une machine booléenne pour représenter (a) des nombres et (b) des instructions sur ces nombres ?

(a) La machine fonctionne avec une logique booléenne (le courant passe ou ne passe pas) : le BIT. C'est très simple : s'il n'y avait que deux nombres, nous conviendrons de dire que si le courant passe (ou si la carte est perforée), cela représente le premier nombre; si le courant ne passe pas (ou si la carte n'est pas perforée), cela représente le deuxième. S'il y avait quatre nombres, il suffit de rajouter une case à la carte : on obtient quatre possibilités, et ainsi de suite. L'informaticien a donc «numéroté les nombres»

(b) Puisqu'on peut numéroter les nombres, il n'y a aucun problème pour numéroter les instructions (1=Additionner, 2=Soustraire, 3=Multiplier, etc...)

Dès lors, derrière la carte perforée se trouve en puissance une représentation univoque d'un état électrique particulier de la machine qui *représente* soit des données, soit des instructions.

Codification alphanumérique et compilateurs

Vint alors *l'alphanumérique*, tant pour les données que pour les instructions, et, avec lui, le début d'une imposture que nous détaillerons par la suite.

(a) Si obtenir un programme qui calcule les salaires est extraordinaire, il serait encore mieux d'avoir un programme qui produise des fiches de salaires personnalisées. Pour cela, il faudrait pouvoir mémoriser non seulement des nombres mais aussi des lettres. Dans les deux cas, c'est le même mécanisme qui va être mis en oeuvre : la numérotation. Nous conviendrons ainsi que A=1, B=2, C=3, ... ,Z=26. On peut même ajouter les signes de ponctuation : virgule = 27, Point = 28, etc.

(b) Nous avions convenu que additionner=1 mais au fur et à mesure que la machine se complique, le nombre d'instructions différentes augmente. Nous conviendrons donc d'écrire dorénavant ADD = Additionner = 1. Le programmeur voit donc sa tâche simplifiée. Il ne doit plus programmer ni en *code* binaire, ni avec des chiffres mais avec des codes mnémoniques. A ce stade, nous parlons de *langage* ASSEMBLEUR. Ce langage nécessite pour être exécuté par la machine d'être traduit en code binaire (le seul que comprend le processeur). Cela peut se faire par l'intermédiaire d'un programme appelé *compilateur*. Le compilateur est le programme informatique qui va traduire ADD en 1, SUB en 2, etc... Ce tournant de l'histoire est fondamental, car l'utilisation de l'alphanumérique est ce qui va permettre *l'illusion informatique*.

Appellation symbolique

Si les nombre 15643 ou 25435 n'évoquent rien de particulier pour l'ensemble des êtres humains, le mot ADD évoque une signification bien spécifique pour quiconque connaît l'anglais. Très rapidement, les langages modernes de programmation permettront de créer des sous-programmes (des parties de programmes) auquel le programmeur donnera un nom qui pour lui est plein de sens. Il est intéressant de constater que ce n'est pas le langage assembleur lui-même qui est exécuté par la machine, mais bien le code machine traduit par l'intermédiaire du compilateur.

Un autre progrès important est l'affectation de noms symboliques affectés à des cases mémoire. Plutôt que de dire «additionner le contenu de la case mémoire 1345 à celui de la case 1789 et le mettre dans la case 1657», il est beaucoup plus commode et significatif d'écrire quelque chose comme

```
case 1345 = SalaireBrut
case 1789 = Prime
case 1657 = SalaireTotal
SalaireTotal = SalaireBrut+Prime
```

En langage Pascal, par exemple, nous pouvons écrire très précisément :

```
Program calcul_salaire;
Var SB, ST, PRIME : INTEGER;
Begin ST := SB + PRIME;
End.
```

La première ligne déclare qu'il faut réserver trois cases en mémoire de taille convenable pour y stocker chaque fois un nombre entier. La deuxième ligne fait le total des deux premières cases mémoire et stocke le résultat dans la troisième[10]. On mesure le progrès considérable qui sépare une telle formulation du langage assembleur. Elle n'en possède pas moins une toute aussi grande rigueur et des règles précises qui permettront à ce programme de déclencher un état électrique précis au bit près de la machine ordinateur.

A ce point, il importe de faire très attention. Un langage de programmation exige une rigueur implacable de la part du programmeur et le respect permanent d'une grammaire impitoyable. Il est en effet totalement indispensable que le compilateur puisse traduire sans ambiguïté les instructions qui lui seront communiquées. Le point derrière le «end» final, par exemple, est tout à fait indispensable. Pas question non plus d'écrire ST = SB + PRIME ou encore PRIME+SB := ST, cela provoquerait une erreur lors de la compilation et aucun programme exécutable ne serait généré. Ainsi, le compilateur n'a pas été conçu pour comprendre le langage humain mais bien une occurrence particulière de celui-ci, occurrence tellement particulière, si rigoureusement structurée et dénué de bon sens, qu'elle en possède un aspect inhumain et cela plus particulièrement lors de l'écriture du programme que lors de sa relecture.

C'est dès ce moment que naît l'imposture. A ce stade, les informaticiens ont, par commodité et à l'aide d'une série d'artifices et de règles précises (pouvant être informatisées), *anthropomorphisé* une manière de communiquer un programme à la machine. Il y a isomorphisme[11] parfait entre un programme écrit dans un langage de programmation et un état électrique particulier et univoque d'une machine ordinateur. A la lecture du code écrit, nous pouvons presque avoir l'impression que la machine nous *comprend*. Il n'en est cependant rien. Nous n'avons fait que travestir de sens des bit et des bytes afin de nous aider à nous rappeler ce que nous souhaitons qu'ils signifient. A la sortie de la machine, nous avons trop tendance à oublier un peu vite que c'est l'homme qui transforme des points lumineux (pixels) allumés sur un écran en lettres, ces lettres en mots, ces mots en paroles et ces paroles en sens.

D'un point de vue conceptuel, même et surtout si cela peut paraître trivial à l'être humain que nous sommes, l'apport principal d'un langage informatique est de permettre de nommer les données et les traitements. Ce nom est une donnée informatique qui fait référence à l'endroit où se trouve(ro)nt en mémoire les bytes décrivant certaines caractéristiques de la données ou les instructions destinées au processeur qui constituent les traitements. D'un point de vue électronique, il n'existe aucune différence concrète entre un ensemble de bytes représentant un chiffre, un nom ou une instruction. Ceci est normal dans la mesure où le principe même du code informatique est la numérotation des chiffres, des noms et des instructions. *L'appellation des variables et des traitements n'a servi qu'à permettre l'isomorphisme entre la numérotation et une représentation, figée, stéréotypée et incomplète d'une partie de la réalité observable.* Les noms que nous avons attribué aux données et variables ne sont pas utilisés dans le programme final communiqué à l'utilisateur et restent à tout jamais dans le code source; la machine n'en a pas besoin pour faire en sorte que le programme fonctionne correctement. Le sens reste au vestiaire, seuls les bits pénètrent dans la machine, seuls les bits en sortent.

Du coté de l'utilisateur : du labyrinthe des modes opératoires aux outils et à la table de travail

Interface fonctionnelle

Tout ordinateur a besoin pour s'animer d'un programme minimal, d'une première instruction. Techniquement, l'ordinateur comprend en mémoire morte[12] un programme de base qui lui permettra d'exécuter d'autres programmes. Le système d'exploitation est le noyau central capable d'exécuter des primitives de bases sur des fichiers (les fichiers contiennent soit des données, soit des programmes). C'est le minimum minimorum pour qu'un utilisateur (qu'il soit ou non informaticien) puisse commencer à travailler.

Pour pouvoir parler au grand golem que constitue le système d'exploitation, il suffit d'inscrire sur son front (le clavier) le nom d'une action qu'il connaît avec un certain nombre de paramètres (sur quoi faut-il exercer l'action et selon quelles modalités). Soit c'est une «formule de vie» et l'ordinateur l'exécute, soit la *formulation* est incorrecte et l'utilisateur est sanctionné par un message tel que «BAD COMMAND». Ce «bad» sans appel et culpabilisant est révélateur d'un climat finalement très émotif qui préside les *rapports* entre l'homme et la machine. Courants sont les messages où l'ordinateur parle à la première personne ou, pire encore, des messages du type «ce client n'existe pas».

Dans les interfaces pour système d'exploitation qui ont présidé l'informatique jusque vers 1985, la formulation-type d'une commande est constituée par le nom d'une action suivi d'un certain nombre de

paramètres, le tout devant être encodé suivant une syntaxe très rigide.

Face à un tel type d'interface, l'utilisateur est confronté au problème suivant. Il ne connaît pas à priori les commandes qu'il faut adresser à l'ordinateur. Celui-ci est prêt à écouter n'importe quelle suite alphanumérique qui sera frappé au clavier. Si cette norme de comportement contribue à entourer la machine d'une aura factice d'intelligence (puisqu'il peut, a priori, tout comprendre) artificielle (puisque cette intelligence émane d'une machine), d'un point de vue de l'hyper-fonctionnalisme[13], elle démontre la bêtise naturelle du concepteur de ce type d'interface : puisque le nombre de commandes possibles est limité, il serait plus intelligent, afin d'éviter des *erreurs* humaines de permettre à l'utilisateur de choisir une commande dans une liste. Au restaurant, on ne demande pas à manger n'importe quoi, on demande le menu et on y sélectionne un plat. Et depuis notre plus jeune âge, nous effectuons un choix en montrant du doigt.

Cette technique va rapidement être mise en oeuvre grâce à un outil qui permet de «montrer du doigt» : la souris. Perçue comme un gadget par la branche dure des partisans de la machine ésotérique elle deviendra rapidement un élément incontournable des machines des années 90.

Plusieurs questions se posent. L'écran est fort petit, il pourrait difficilement contenir toutes les commandes possibles. En outre, comment les présenter ? Des réponses seront apportées. D'une part, les commandes sont groupées par titre de chapitre (comme au restaurant : hors d'oeuvre, plats, desserts, ...). Sur l'écran seuls sont proposés les titres de chapitre dans une *barre de menu* située en haut de l'écran. Un clic de souris sur une tête de chapitre provoque le déroulement du menu à l'écran. Un clic de souris sur la commande désirée permet l'exécution de cette commande.

D'autres questions se posent au niveau des paramètres. Pourrait-on imaginer de présenter une liste exhaustive de tous les fichiers ? Comment distinguer les fichiers de données des fichiers programmes ? Comment distinguer entre eux les fichiers de données suivant qu'ils sont des documents de traitement de texte, des documents produits par un tableur, etc. ? La représentation des fichiers se fera sous forme de petits dessins appelés icônes. Il s'agit de petits logos représentant symboliquement le fichier sous une forme graphique. Là encore, l'utilisateur peut disposer ses fichiers dans divers dossiers qui seront à leur tour inclus dans d'autres dossiers, etc.

Corollairement à cela, ce nouveau type d'interface inverse l'ordre de dialogue avec l'utilisateur. Avec l'interface précédente, l'utilisateur communiquait d'abord l'action qu'il voulait accomplir et ensuite, via les paramètres, ce sur quoi il voulait agir. Avec la nouvelle interface, on choisit tout d'abord l'objet sur lequel on veut agir et ensuite l'opération que l'on désire effectuer. Ce qui est vrai au niveau du système d'exploitation l'est encore au niveau d'un traitement de texte. On sélectionne d'abord à l'aide de la souris un groupe de caractères (en le noircissant), on choisit ensuite via le menu l'opération que l'on veut exécuter (souligner, effacer, agrandir, etc...).

Non procéduralité

Revenons quelques instants sur notre exemple du calcul des racines d'une équation. L'interface originelle de ce genre de programme aurait imposé un développement linéaire du programme. Celui-ci aurait demandé la valeur de A, ensuite la valeur de B et enfin la valeur de C. Au terme de ces trois entrées, l'ordinateur aurait communiqué la valeur des racines. Le déroulement du programme est séquentiel. Impossible de revenir en arrière en cas d'erreur. Si nous voulons tester 10 valeurs de C pour deux mêmes valeurs de A et B, il faut effectuer 30 entrées numériques dont 18 redondantes.

Dès les années 80, le gestionnaire d'écran permettra de changer cette procéduralité des entrées. L'utilisateur a devant lui un écran qui ne défile pas. Au déplacement de l'écran et au curseur fixe, se substituent un écran fixe et un curseur mobile que l'utilisateur peut déplacer d'une case à l'autre. Ce n'est qu'au moment de la validation de l'écran que le calcul des racines s'effectuera. Si on ajoute à ce gestionnaire d'écran une barre de menu, on obtient une différence notable : l'utilisateur a de plus en plus le pouvoir de prendre des initiatives. Un programme n'est plus tant une suite d'opérations à effectuer dans un ordre déterminé qu'une palette d'outils que l'utilisateur peut utiliser quant il le désire. Avec les perfectionnements techniques, il est même devenu possible de passer d'un programme à l'autre sans devoir nécessairement clôturer le travail en cours. Par exemple, on peut quitter provisoirement un tableur, consulter une base de données, puis reprendre le travail avec le tableur dans l'état où on l'avait laissé.

Communication de données entre applications

Dès lors, la nécessité de pouvoir facilement échanger des données entre plusieurs programmes se fait directement sentir. Ou, mieux encore, de partager certaines données entre plusieurs applications et/ou plusieurs utilisateurs. On peut, par exemple insérer un tableau de chiffres conçu avec un tableur dans un rapport conçu avec un traitement de texte. Le fait de modifier par la suite la feuille de calcul ad hoc dans le tableur aura pour effet de modifier instantanément le tableau inséré dans le rapport.

A ce stade, nous avons détaillé le cycle de vie d'un programme, tentant d'en souligner le caractère spéculatif et inventif. Afin de communiquer un programme à la machine ordinateur, les informaticiens se sont dotés d'un code qui est devenu au fil des temps un langage dans la mesure où il permet d'inventer de nouveaux mots, pour nommer les traitements et les données.

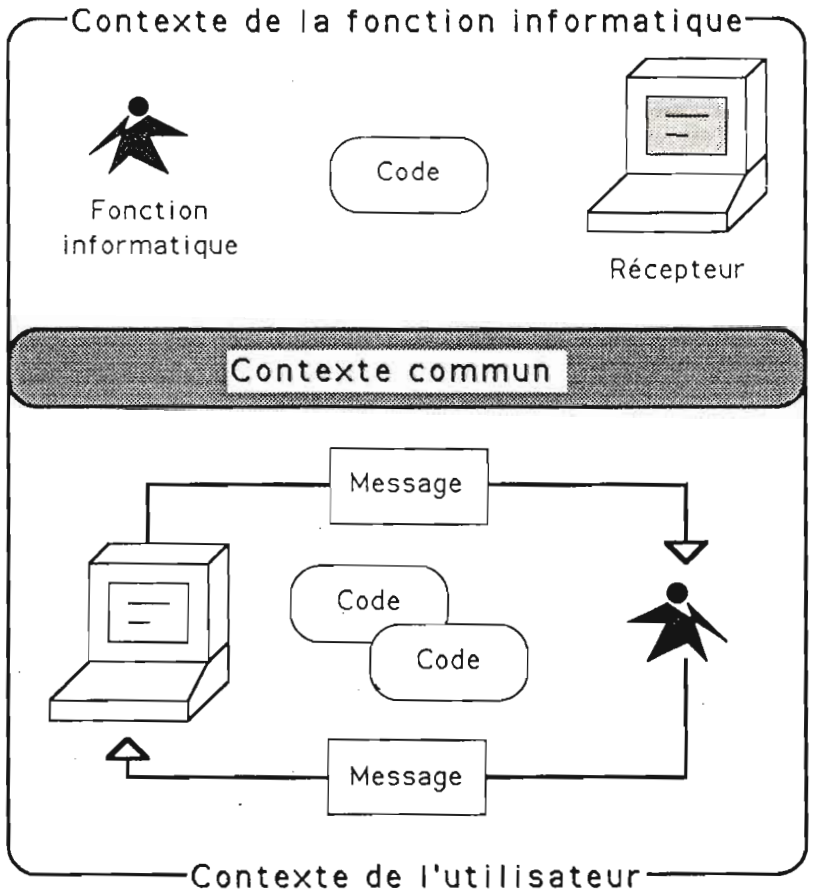
De son coté, au fil des temps, le comportement de l'utilisateur a évolué vers une autonomie plus grande. L'utilisateur d'informatique ne répond plus passivement aux questions de l'ordinateur mais peut prendre l'initiative d'organiser et de découper son travail comme il l'entend. Un programme d'ordinateur est de moins en moins un labyrinthe complexe où, à chaque carrefour, l'utilisateur doit indiquer son choix d'une manière précise. L'ordinateur est devenu une table de travail, et les programmes autant d'outils spécifiques, souples et complémentaires qui permettent de créer ou modifier l'information. De plus en plus l'utilisateur peut prendre l'initiative.

De la dualité contextuelle et des moyens d'y remédier

La dualité contextuelle

Si nous mettons ce dialogue homme-machine/machine-homme que produit l'informatique en regard d'un processus de communication entre deux individus, nous nous trouvons en présence d'un processus de communication *à deux étages*. Dans le premier niveau le programmeur communique à la machine les règles d'un dialogue futur avec un utilisateur quelconque; dans le deuxième niveau, la machine dialogue avec l'utilisateur en écoutant par l'intermédiaire d'un clavier, d'une souris et en émettant des messages stéréotypés sur un écran vidéo.

Par rapport au schéma classique de la communication tel que Jacobson[14] le proposa en 1963, nous constatons que ces deux étages de la communication posent un problème fondamental que nous nom-



merons «dualité contextuelle» : le contexte où s'élabore le dialogue entre l'informaticien et la machine et, plus tard, celui entre la machine et l'utilisateur sont en partie disjoints. Dans le cas de fabrication de logiciels à usage général (traitement de texte, tableurs, langages de gestion de base de données, ...), l'informaticien écrit des programmes pour des utilisateurs qu'il n'a jamais rencontrés et qu'il ne rencontrera jamais. Il choisit lui-même les modes opératoires que l'utilisateur devra mettre en oeuvre. Ce choix du code est opaque pour l'utilisateur mais n'est pas fortuit : il puise ses racines dans une histoire, celle, collective, de l'informatique et celle, individuelle, de l'informaticien. Il se heurte à une histoire différente, inconnue et singulière, de l'utilisateur. Nous illustrerons ce problème de dualité contextuelle par deux exemples.

Le premier exemple, bien qu'étant plus lié au développement technologique en général qu'à celui de l'informatique en particulier nous semble suffisamment révélateur pour mériter d'être explicité. Il s'agit d'un outil que nous employons tous les jours : l'ascenseur. Lequel d'entre nous n'a pas été contrarié par l'arrêt inopiné d'un ascenseur sans que la personne qui l'aurait appelé soit présente devant les portes béantes ? Qu'en pensons-nous ? Soit que la technologie n'est pas au point, soit que la personne ayant effectué l'appel, lassée d'attendre a pris l'escalier. Evidement, c'est toujours aux heures d'affluence que la machine commet ce genre de bizarreries...

Tout comme l'utilisateur pour comprendre un programme doit, autant que faire se peut, se mettre dans la peau de celui qui l'a conçu, tentons de retracer le raisonnement du constructeur de l'ascenseur à travers une diachronie volontairement simplifiée. Il y eut tout d'abord le monte-charge. Ensuite, quelqu'un a eu l'idée d'adapter cette technique au transport de personnes. Dans les hôtels chics du début du siècle on peut encore admirer quelques uns de ces spécimens nécessitant la présence d'un opérateur humain spécialisé. Petit à petit, on vit cependant disparaître le «groom» qui céda la place à un clavier on ne peut plus simple. Actuellement, il suffit pour l'utilisateur d'appuyer sur le bouton correspondant à l'étage où il désire se rendre. C'est du coté de *l'appel* de l'ascenseur que tout se complique. Le concepteur de l'interface d'appel de l'ascenseur est confronté au problème suivant : il faut éviter d'arrêter un ascenseur qui descend pour quelqu'un qui désire monter et vice-versa, et ce pour deux raisons. Tout d'abord, pour éviter des arrêts intempestifs aux personnes qui veulent descendre; ensuite parce que la personne qui monte n'a pas tellement envie de descendre d'abord pour remonter ensuite. S'il n'y qu'un seul bouton d'appel, il est impossible de savoir si celui qui appuie dessus désire monter ou descendre. Le concepteur a donc l'idée d'installer deux boutons : l'un avec une flèche vers le haut, que l'utilisateur pressera (fort intuitivement se dit le concepteur) s'il désire monter, l'autre avec une flèche vers le bas, pour l'utilisateur qui désire descendre. Comme à un même étage peuvent se trouver une personne qui veut monter et une autre qui désire descendre, le dispositif est conçu de manière à pouvoir enregistrer *deux* appels distincts. Malheureusement, cette flèche est bien souvent interprétée d'une manière différente.

Imaginons la situation suivante. L'ascenseur se trouve au 20ème, chargé d'un groupe de personnes descendant au rez-de chaussée.

Une personne, qui se trouve au 19ème étage, désire monter au 20ème. Elle observe, sur le tableau de contrôle, que l'ascenseur se trouve plus haut, et appuie sur la flèche vers le bas, signifiant par là qu'elle désire que *l'ascenseur* descende. Celui-ci s'arrête au 19ème, embarque la personne, puis descend au rez-de-chaussée avant de remonter au 20ème étage. Notre utilisateur aura donc parcouru 39 étages (19 vers le bas et 20 vers le haut) au lieu d'un seul nécessaire. A la remontée, l'ascenseur embarquera probablement d'autres personnes qui, voulant *descendre* et voyant sur le tableau de contrôle que l'ascenseur était au rez-de-chaussée, ont appuyé sur la flèche vers le haut *pour le faire monter*.

En l'absence d'un panneau de contrôle, cette même situation est encore plus catastrophique. Ne sachant si l'ascenseur est plus haut ou plus bas que lui, l'utilisateur appuie sur les deux boutons. Le système enregistre deux appels. L'ascenseur s'arrête et éteint la flèche vers le bas. Il mémorise toutefois qu'il reste un appel d'une personne qui désire monter. A la remontée, il s'arrête au 19ème, irritant celui qui s'y trouvait par un arrêt pour un fantôme.

Si par contre l'utilisateur connaît l'interprétation faite par le système de la flèche vers le haut et si les portes sont aveugles, il peut être énervé en entendant l'ascenseur passer sans s'arrêter, surtout s'il est pressé. Dans ce cas, il appuie sur tous les boutons qui passent à sa portée, ce qui ne fait qu'augmenter les arrêts intempestifs et énerver les autres utilisateurs. Cela explique que le nombre d'arrêts *bizarres* augmente avec l'utilisation de l'ascenseur et donc en période d'affluence.

Voici donc un exemple qui illustre à merveille ce problème de dualité contextuelle. Le concepteur des boutons d'appels de l'ascenseur associe à une touche une signification qui lui semble évidente puisqu'elle concerne le *désir de l'utilisateur* (monter ou descendre) et l'utilisateur donne une signification contradictoire mais tout aussi intuitive qui concerne en fait le *mouvement souhaité de l'ascenseur*.

Notre deuxième exemple est tiré d'une formation au traitement de texte. Les utilisateurs les plus durs à «convertir» seront probablement ceux qui depuis

des années travaillent sur une machine électrique ou mécanique. Combien de dizaines de milliers de fois n'ont-ils pas tapé la lettre «O» à la place du chiffre zéro. Parce que dans l'esprit d'une dactylo seule la trace visible sur le papier compte. En informatique, il n'en est rien : la seule chose importante est la connexion électrique qui se situe en amont de la touche et, pour un ordinateur, il n'y a pas plus de ressemblance entre la lettre O et le chiffre 0 qu'entre un 6 et un "A". Rien n'empêcherait, sur le plan technique, que l'informaticien apprenne à la machine de confondre la lettre 0 avec le zéro dans un certain nombre de cas pertinents (lors de la lecture d'un nombre par exemple). Cette dualité contextuelle qui mine la communication entre l'informaticien et l'utilisateur par machine interposée est le problème central que les informaticiens tentent de résoudre depuis toujours. On peut aujourd'hui distinguer trois types de solutions : *la standardisation, l'évolution de la puissance des langages et la métaphore.* A notre sens, il est fondamental de percevoir dans ces trois éléments, qui forment la clé de voûte du développement informatique de cette dernière décennie, la volonté des informaticiens de résoudre un problème fondamental de communication.

Les remèdes à la dualité conceptuelle

La standardisation

Principe

L'idée source de la standardisation est la suivante. Puisqu'il existe un problème de code entre l'utilisateur du programme et son créateur, convenons d'un certains nombre de modes opératoires stéréotypés applicables dans un nombre limité de situation typiques.

Exemples

A l'origine la touche «return» était associée au mouvement mécanique du *retour* du chariot de la machine à écrire. Petit à petit elle est devenue une convention pour marquer la fin de l'écriture au clavier d'une information formant un tout. Elle *signifie* quelque chose comme «j'ai fini de taper ce que je devais taper, passons au stade suivant».

La touche de tabulation (marquée «TAB» ou encore «—>|») permettait à l'utilisateur d'avancer de plusieurs espaces suivant la position de certains taquets réglable. Sur les machines à écrire plus récentes, une touche permettait de positionner un taquet à la position courante de la tête d'écriture, une autre touche permettait de l'enlever. L'utilisateur pouvait ainsi construire des colonnes invisibles sur sa feuille et passer de l'une à l'autre par simple pression d'une touche, l'alignement était identique d'une ligne à l'autre. Sur la plupart des programmes, la touche de tabulation permet de passer dans la colonne suivante. Elle *signifie* quelque chose comme «passons à la case suivante que je dois remplir».

Une autre touche appelé ESC (escape; évasion en anglais) permet en général d'annuler une opération en cours pour revenir au niveau supérieur. Typiquement, cette touche permet à l'utilisateur de revenir au menu principal d'un programme à partir d'un sous menu.

Avantages

Petit à petit, certaines touches du clavier acquièrent une *signification.* Cette signification naît de l'usage et d'un «effet boule de neige» : les concepteurs de programmes ont tout intérêt à utiliser des conventions qui correspondent aux pratiques en cours et donc à renforcer ces pratiques. L'utilisateur a évidement tout à gagner dans une uniformisation des modes opératoires des différents logiciels. Par ce biais, il n'est en effet plus nécessaire de réapprendre une série de mode opératoires courants qui restent constants d'un logiciel à l'autre.

L'évolution de la puissance des langages

Principe : la notion de primitive

On appelle primitive une instruction exécutable par un ordinateur, sans qu'elle soit décomposable en séquences significatives par l'utilisateur. Au tout début de l'informatique, la seule primitive possible était de brancher ou de débrancher un fil électrique. Depuis lors, les choses ont bien évolué par l'intermédiaire d'une amélioration de *l'interface* entre l'homme et la machine. L'interface est le lieu de passage obligé entre la machine et le monde extérieur. Elle ne se limite pas aux seuls aspects physiques (clavier, écran, imprimante) mais incorpore également un certain nombre de règles de communication. On peut dès lors parler de convivialité ou d'ergonomie. C'est ce qu'on appelle le système d'exploitation (e.g. le D.O.S. sur P.C., le Finder sur Macintosh, Unix sur certains mini, etc...). La particularité de l'interface est de «traduire» une seule instruction de l'utilisateur en une série considérable d'instructions directement exécutables par le processeur de l'ordinateur. Ainsi, par exemple, en quelques touches (ou en quelques coups de souris), l'utilisateur peut donner l'ordre via l'interface d'effacer un fichier. Cet effacement d'un fichier se concrétise par l'exécution de plusieurs milliers d'instructions par le processeur central. L'instruction de base au niveau du processeur (par exemple : imprimer un caractère à l'écran, faire la somme de deux nombres, lire la touche que l'utilisateur a frappé au clavier, etc...) n'a guère évolué dans le temps en comparaison des primitives mises à disposition de l'utilisateur. Le fossé entre les deux a été comblé par des progrès de l'interface.

Exemples

Afin de mieux nous faire comprendre, prenons un exemple de la vie courante : l'histoire des cassettes. Au début, il y eut des bandes magnétiques. Ces bandes étaient composées d'un rouleau autour duquel était enroulé un support plastique métallisé souple et solide. Ce ruban passait devant une tête magnétique et allait ensuite s'enrouler sur une bobine vide. A la fin de l'écoute, la bobine pleine était vide tandis que la bobine vide était remplie. Un premier perfectionnement fut d'enregistrer sur la moitié de la bande (dans le sens de la largeur), la tête se trouvant non plus au centre de la bande, mais dans la partie (la «piste») inférieure de celle-ci. Cette technique permettait d'enregistrer une durée deux fois supérieure sur la même longueur de bande. Elle supposait que l'utilisateur retourne la bande pour pouvoir l'écouter entièrement. Ensuite apparut la cassette qui garde le même principe mais qui intègre deux bobines dans un support en plastique. Actuellement il existe des modèles «autoreverse» qui effectuent une lecture de la bande sans devoir retourner la cassette (c'est la tête elle-même qui se retourne). Sur les cassettophones de voiture, l'insertion d'une cassette dans l'appareil suffit généralement à mettre celui-ci en route (afin de minimiser le nombre de mouvements distrayants du conducteur). Examinons l'évolution des primitives (commandes) nécessaires pour écouter deux faces d'une bande.

Evolution des primitives

Enregistreur à bande

1. Prendre la bobine et la fixer sur le plateau de gauche.
2. Fixer une bobine vide sur le plateau de droite
3. Saisir l'extrémité de la bande et la faire passer devant la tête de lecture
4. La fixer à la bobine vide
5. Tourner celle-ci de façon à tendre la bande
6. Appuyer sur la touche play...
7. A la fin de la bande, inverser les deux bobines
8. Saisir l'extrémité de la bande et la faire passer devant la tête de lecture.
9. La fixer à la bobine vide
10. Tourner celle-ci de façon à tendre la bande
11. Appuyer sur la touche play...

Cassettophone non «autoreverse»

1. Insérer la cassette dans son logement
2. Appuyer sur la touche play
3. A la fin de la cassette, extraire celle-ci de son logement
4. La retourner et la remettre dans son logement
5. Appuyer sur la touche play

Cassettophone «autoreverse»

1. Insérer la cassette dans son logement
2. Appuyer sur la touche play

(à la fin de 1ère face, la tête se retourne et l'appareil lit la 2ème face)

Cassettophone «autoreverse» pour voiture

1. Insérer la cassette dans son logement

Ainsi donc, le nombre de primitives pour manipuler une bande magnétique a considérablement diminué, mais leur puissance a augmenté. Aujourd'hui, il suffit d'un seul geste simple avec un cassettophone autoreverse pour écouter les deux faces d'une bande, travail qui nécessitait auparavant plus d'une dizaine d'opérations relativement délicates. La formation que l'on devrait fournir à un utilisateur désireux d'écouter les deux faces d'une cassette se réduit à peu de choses dans le dernier des cas. Il suffirait de lui montrer le logement où il faut introduire la cassette et de lui expliquer qu'une cassette est un dispositif qui enregistre de la musique. Cette formation est alors purement opératoire. Elle n'explique pas ce qu'est une cassette, elle détaille les gestes à accomplir pour pouvoir s'en servir. A la limite l'auditeur n'a même pas besoin de savoir qu'il y a deux faces.

De la même manière, il fallait aux premiers jours de l'informatique plusieurs mois de travail de travail et de connaissances techniques de haut niveau pour mettre au point un programme simple de gestion d'adresses. Aujourd'hui, un utilisateur moyen peut le réaliser en quelques jours sur micro ordinateur. Semblablement, les premiers éditeurs de texte permettaient de travailler le texte ligne par ligne, en les numérotant. Pour visualiser une page de texte, il fallait donner la commande «imprimer à l'écran de la ligne n°X à la ligne n°Y». Aujourd'hui, n'importe quel éditeur de texte à moitié moderne travaille en plein écran et permet en une pression de touche de visualiser la page précédente, la page suivante, etc...

Avantages et inconvénients

L'énorme avantage de cette augmentation de puissance des commandes consiste en une augmentation du confort pour l'utilisateur. Le code des commandes est moins compliqué à retenir et il y a moyen de faire de plus en plus de choses en donnant de moins en moins d'instructions. Le risque d'erreur de manipulation s'en trouve aussi nettement diminué. Le gros inconvénient de cette tendance est l'augmentation de *l'opacité* du système. Nul n'est besoin aujourd'hui de savoir qu'une cassette contient une bande magnétique pour pouvoir écouter de la musique.

La métaphore

Principe

Puisqu'il existe un problème de décalage entre le contexte de l'informaticien et celui de l'utilisateur, il suffit de présenter les commandes comme des opérations sur des métaphores universellement connues.

Exemple

L'exemple le plus connu est probablement le «bureau» de l'Apple Macintosh. Chaque fichier est représenté par une icône sur l'écran. Ces icônes ont toutes la même taille mais représentent des dessins différents caractéristiques des différents programmes ou documents qu'elles symbolisent. Pour effacer un fichier, l'utilisateur doit simplement saisir l'icône à l'aide d'une souris et la faire glisser vers un dessin de corbeille à papier. Les primitives présentes via une interface graphique ou une interface classique comme celle du D.O.S. sur PC sont tout à fait similaires.

Avantages et inconvénients

L'avantage de ce type de méthode est que l'utilisateur ne se trouve pas dépaysé. Il reste dans son univers familier. Mais la métaphore possède aussi des limites. Il semblerait qu'un utilisateur moyen apprend plus rapidement avec ce type d'interface. Par contre, après une certaine accoutumance, beaucoup d'utilisateurs se plaignent de son coté opaque, peu transparent.

Quelques lignes de forces d'une formation au traitement de texte

On ne peut comprendre un programme sans comprendre l'informatique. On ne peut comprendre l'informatique sans un exposé comme celui effectué ci-dessus. Cette formation précède nécessairement la formation à un programme particulier. On imagine mal, en effet, que l'on puisse parler d'une voiture particulière sans parler d'abord de l'automobile en général.

Que faire lors de la formation à un programme particulier, par exemple un traitement de texte ? Tout d'abord, la formation générale doit avoir abordé en profondeur les règles d'interface utilisées sur un type particulier de machine. Ces règles (et c'est une évolution récente et permanente) tendent à s'uniformiser pour tous les types de programmes et pour tous les types de machines.

Nous avons montré plus haut que le travail de la fonction informatique ne se limite pas à la programmation à l'aide d'un langage symbolique, mais qu'elle constitue une démarche communicative en quatre temps effectuée par la médiation d'une machine. Cette partie de la genèse d'un programme n'est que la partie visible et étincelante de l'iceberg. Or, classiquement, c'est elle qui est communiquée à l'utilisateur, et uniquement dans sa composante code-objet. Les 7/8èmes de l'iceberg restent donc immergés.

Dans le canevas de formation que nous proposons ci-après, nous croyons qu'il est important de restituer à l'élève, dans les grandes lignes, la démarche informatisante dans les trois des quatre phases que nous avons détaillé (analyse de l'existant, analyse fonctionnelle, codage et choix de modes opératoires). Dans un but pédagogique, nous partirons du plus simple pour aller vers le plus compliqué même si la démarche d'analyse de l'existant doit nécessairement partir de l'extrême complexité de la réalité pour en extraire un certain nombre de concepts.

Soucieux de ne pas alourdir outre mesure cet exposé, nous nous focaliserons sur la notion de *paragraphe*.

L'analyse de l'écriture

Avant même de mettre en oeuvre un programme de formation au traitement de texte, nous voyons deux idées fausses, couramment répandues dans le grand public qu'il importe de détruire.

Ce qui est vrai pour de nombreux logiciels l'est ici aussi : «on» a l'impression qu'il est facile d'apprendre tout seul un traitement de texte. De plus en plus, les auteurs de tels logiciels joignent d'ailleurs à leurs disquettes les uns une aide «on-line», les autres un manuel d'apprentissage ou un manuel de référence, les derniers enfin un «tutorial», petit cours d'EAO. Nous sommes convaincus qu'il est facile d'apprendre *vite et très mal* les quelques commandes nécessaires pour produire et imprimer un document présentable. L'apprentissage de manière autodidacte d'un traitement de texte peut amener à des conduites-réflexes difficilement remises en question par la suite. Cet apprentissage par essais et erreurs est d'ailleurs reconnu comme un processus très onéreux pour l'entreprise.

Une deuxième idée que l'on rencontre fréquemment est celle suivant laquelle le remplacement d'une machine à écrire par une machine de traitement de texte «ne change rien», tout comme on pourrait croire naïvement que l'informatisation d'une entreprise ne modifie pas en profondeur le fonctionnement de celle-ci. Pourtant nous croyons qu'elle amène des bouleversements fondamentaux.

Tout d'abord c'est le *statut* même *de l'écriture* qui en est bouleversé. Sur une machine à écrire, il n'y a pas moyen de supprimer ou de rajouter un paragraphe car cela nécessiterait la refrappe de tout le texte en

amont. Avec un traitement de texte, cela est non seulement possible[15] mais aussi facile. Cela signifie que les habitudes de travail risquent de se modifier. Jadis un texte était envoyé «à la frappe» lorsqu'il était dans sa forme définitive, aujourd'hui il peut être frappé dès son élaboration et modifié des centaines de fois, l'ordinateur se chargeant instantanément de refaire les décalages nécessaires afin d'assurer une mise en page correcte. Le texte n'est donc plus, dans sa forme, une suite de mots écrits une fois pour toutes, mais grâce à l'informatique un ensemble de signes altérables et améliorables à l'infini.

Une deuxième modification importante porte sur l'archivage des documents. Jadis celui-ci se faisait sous la forme d'une reproduction de l'écrit dans sa forme définitive, sous forme de carbone, photocopie, micro film, etc. Aujourd'hui l'écrit sera archivé le plus souvent sous forme magnétique (disquettes, bandes magnétiques, ...). Cela pose un double problème d'authenticité des données (puisque celles-ci sont malléables à merci) et de fragilité (les disquettes sont sensibles à la chaleur et aux perturbations du champ magnétique). C'est pour cela que l'on voit apparaître peu à peu des disques optiques qui permettent une seule écriture et un nombre infini de lectures[16]. Une formation à l'informatisation de la frappe de documents ne se résume pas à apprendre un traitement de texte mais doit aussi, par exemple, apprendre à faire des sauvetages réguliers du disque dur qui est un support de grande capacité mais reste fragile.

Une troisième mutation porte sur l'exigence d'une finition supérieure. On exigera facilement d'un agent manipulant un traitement de texte un écrit sans ratures, bien cadré, justifié à droite et à gauche, etc., ce qui paraît difficile avec une machine à écrire manuelle, même électrique.

Ainsi, la formation d'un utilisateur au traitement de texte devrait aussi viser à l'aider à remettre en question quelques modes opératoires profondément ancrés dans ses habitudes de travail, à gérer les multiples versions d'un même document et à archiver d'une manière intelligente.

Décomposition en concepts universels

L'informatique n'a pas inventé le traitement du texte. Les concepts présents dans tout texte ne sont guère récents et existaient d'une manière ou l'autre depuis de nombreux siècles, inscrits sur des tablettes d'argile, gravés dans la pierre, écrits sur papyrus ou imprimés sur papier. Ainsi nous pouvons parler, en partant du plus simple pour aboutir au plus complexe, de caractères, de mots, de lignes, de paragraphes, de pages et de textes. L'informaticien, par la médiation d'un programme de traitement de texte, met à la disposition d'un utilisateur des opérations types permettant de les modifier à façon. Cette division successive d'une réalité complexe (ici le texte) en «objets» de plus en plus simples et malléables est typique de la démarche informatique. Nous remarquons que ces six concepts sont imbriqués les uns dans les autres :

- Un mot se compose d'un ou plusieurs caractères.
- Une ligne comprend zéro, un ou plusieurs mots.
- Une ou plusieurs lignes constituent un paragraphe.
- Une page est formée d'un ou plusieurs paragraphes.
- Une ou plusieurs pages constituent un texte.

Cette imbrication peut donner lieu à une confusion de concepts. Ainsi supposons que notre premier texte soit le mot bonjour écrit sur une page. Ce mot est à lui seul ligne, paragraphe, page et texte. Cette imbrication implique que certaines opérations possibles sur un caractère le sont aussi sur un mot, une ligne, un paragraphe, une page, un texte en tant qu'ensemble de caractères, etc.

Propriétés de ces concepts

Passons rapidement en revue les caractéristiques des concepts détaillés ci-dessus.

Le *caractère* comporte plusieurs caractéristiques : sa police, sa taille, sa mise en forme (**gras**, souligné, *italique*., etc...). Il importe de bien distinguer ces trois niveaux indépendants de mise en forme du caractère. Ils peuvent d'ailleurs être combinés à volonté. Si le caractère appartient à une et une seule police de caractères, s'il possède une taille donnée, il peut néanmoins subir plusieurs effets de mise en forme. Par exemple, on pourrait imaginer d'avoir un caractère simultanément gras, italique et souligné.

Le *mot* est un ensemble de lettre commençant et se terminant par un caractère qui n'est pas une lettre et qu'on appelle séparateur. Le séparateur le plus couramment utilisé est l'espace blanc mais on rencontre fréquemment d'autres séparateurs : ; ?, . etc.

La *ligne* est une suite de mots qui occupe la même hauteur sur le papier.

Le *paragraphe* est une suite de lignes terminée par un caractère de contrôle que nous appelerons fin de paragraphe ou «return». Un paragraphe possède une caractéristique importante : il peut être centré, aligné à gauche, aligné à droite, aligné à gauche *et* à droite (on dit alors justifié). Il possède une marge à droite et à gauche ainsi qu'un interlignage, etc.

Une confusion courante apparaît entre les notions de paragraphe et de ligne. A notre sens, cette confusion trouve son origine dans le rôle double joué par le même levier de *retour*[17] de chariot sur les machines à écrire. Ce levier prit la forme d'une touche sur les machines à écrire électriques. On actionne ce levier pour deux raisons différentes : soit la *ligne* est trop courte et il ne reste plus suffisamment d'espace pour inscrire lisiblement les mots qu'il reste à frapper, soit

l'auteur désire aller à la ligne pour marquer la fin d'une idée et d'un paragraphe.

Sur un traitement de texte, la machine se charge elle-même de la première opération mais oblige l'utilisateur à accomplir la seconde. Dans le premier cas, en effet, elle est capable de se rendre compte que l'utilisateur est à court d'espace; par contre, elle ne peut deviner qu'il désire commencer une nouvelle idée en marquant une rupture dans le texte. Cette solution présente un avantage considérable : lorsqu'on ajoute un mot dans un paragraphe, la machine se charge de réarranger les mots dans les différentes lignes en évitant les lignes creuses. Cette manière de faire permet de taper des centaines de caractères sans se soucier de savoir quand aller à la ligne (la «frappe au kilomètre»).

Une erreur commune chez les utilisateurs ayant l'habitude de la machine à écrire et qui abordent le traitement de texte consiste à appuyer sur la touche <return> (qu'ils assimilent à une nouvelle ligne) avant la fin de la ligne. Lors d'une première frappe, ceci ne pose pas de problème. Le principe du traitement de texte étant la modification fréquente du document, cette attitude crée cependant des catastrophes dans la mesure ou l'ajout de quelques mots dans un paragraphe a pour effet de modifier l'endroit de tous les sauts de lignes successifs à cet ajout.

Nous nous trouvons ici en plein dans un problème de dualité contextuelle. Pour la dactylo, la touche <return> est instinctivement liée à un *retour* de chariot. Pour l'informaticien, la dactylo est censée frapper au kilomètre et n'utiliser la touche <return> que pour indiquer le début d'un nouveau paragraphe.

Dans une formation durable ce problème vient d'être résolu comme suit :

- *compréhension du concept de paragraphe analysé par l'informaticien*
- *mise en évidence d'une dualité contextuelle (pour la dactylo : ligne = paragraphe)*
- *élimination de cette dualité contextuelle par la mise en évidence du nouveau concept de paragraphe inventé par l'informaticien et par les avantages de cette nouvelle conceptualisation de la notion de paragraphe par rapport à l'ancienne.*

Les primitives exécutables sur ces concepts

Les primitives permettent de modifier les caractéristiques des concepts mis en évidence ci-dessus.

Il importe, a ce stade, de distinguer les opérations que l'on peut effectuer sur un paragraphe des modes opératoires, i.e. des séquences de touches nécessaire pour y arriver, qui seront brièvement évoqués au point suivant.

En ce qui concerne un paragraphe, nous pourrons modifier son alignement (gauche, droite, centré, justifié) ainsi que sa largeur (marge de droite et marge de gauche), son interlignage (1, 1$^1/_2$, 2). Classiquement, il sera aussi possible de supprimer un paragraphe ou de le déplacer ailleurs dans le texte sans devoir le retaper, de le dupliquer, etc...

Suivant les traitements de textes, d'autres primitives pourront être envisagées. Ainsi, il sera possible de faire en sorte que le paragraphe soit toujours précédé d'un saut de page (intéressant si celui-ci est un grand titre de chapitre, par exemple); de faire en sorte que les lignes d'un paragraphe soient «solidaires» (dans ce cas, le paragraphe ne sera jamais coupé par un saut de page); etc...

> *«Former à l'informatique... c'est refaire dans les grandes lignes, et en se basant autant que possible sur l'univers de l'utilisateur, l'analyse du réel telle qu'a dû l'effectuer l'informaticien qui propose un programme par machine interposée.»*

Les modes opératoires

Ce stade opératoire intervient toujours après la première phase d'analyse. A ce stade, nous pensons qu'il est bon de distinguer deux sortes de commandes : les commandes de déplacement dans le texte et les commandes de modification du texte. Signalons un élément important présent dans tout logiciel. Il s'agit du curseur. Ce curseur indique «*l'endroit où l'utilisateur se trouve*». C'est la place où ira se mettre le prochain caractère tapé au clavier. Le curseur est symbolisé par un clignotement à l'écran.

Nous n'entrerons pas dans le détail des commandes, celles-ci variant selon la marque du traitement de texte et selon le type de matériel choisi. Nous tenons cependant à souligner que la démarche décrite ci-dessus reste valable pour tout traitement de texte.

Résumons-nous.

Nous avons voulu montrer que l'informatique est d'abord une démarche avant d'être une technologie. L'ordinateur est le support technologique concret qui rend cette démarche possible. Le travail de l'informaticien est avant tout un travail de communication en fonction duquel la technologie s'est peu à peu modifiée au fil des temps. De cette activité de l'informaticien, le grand public ne voit bien souvent que le seul aspect de la programmation. Et pourtant celui-ci passe la moitié du temps à analyser et à tenter de

comprendre en profondeur ce qu'il doit «mettre en programme».

Former à l'informatique, ce n'est donc pas pour nous d'abord former à la technique, mais c'est refaire, dans les grandes lignes, et en se basant autant que possible sur l'univers de l'utilisateur, l'analyse du réel telle qu'a dû l'effectuer l'informaticien qui propose un programme par machine interposée. Cette analyse ne fait que mettre en lumière le choix d'une nouvelle structuration d'une réalité ancienne que l'utilisateur adulte connaît intuitivement en général fort bien pour l'avoir pratiquée durant longtemps. Une fois que ce stade est dépassé, il devient possible de montrer les artifices techniques particuliers utilisés par un programme déterminé afin d'aider l'utilisateur à accomplir plus facilement les tâches anciennes.

Cette approche a pour but de développer une compétence générale du traitement de texte dans le sens de la compétence linguistique[18]. Bien plus que l'apprentissage ponctuel d'un traitement de texte éphémère, elle vise à rendre l'utilisateur capable d'apprendre par lui-même n'importe quel traitement de texte qu'il n'a jamais rencontré.

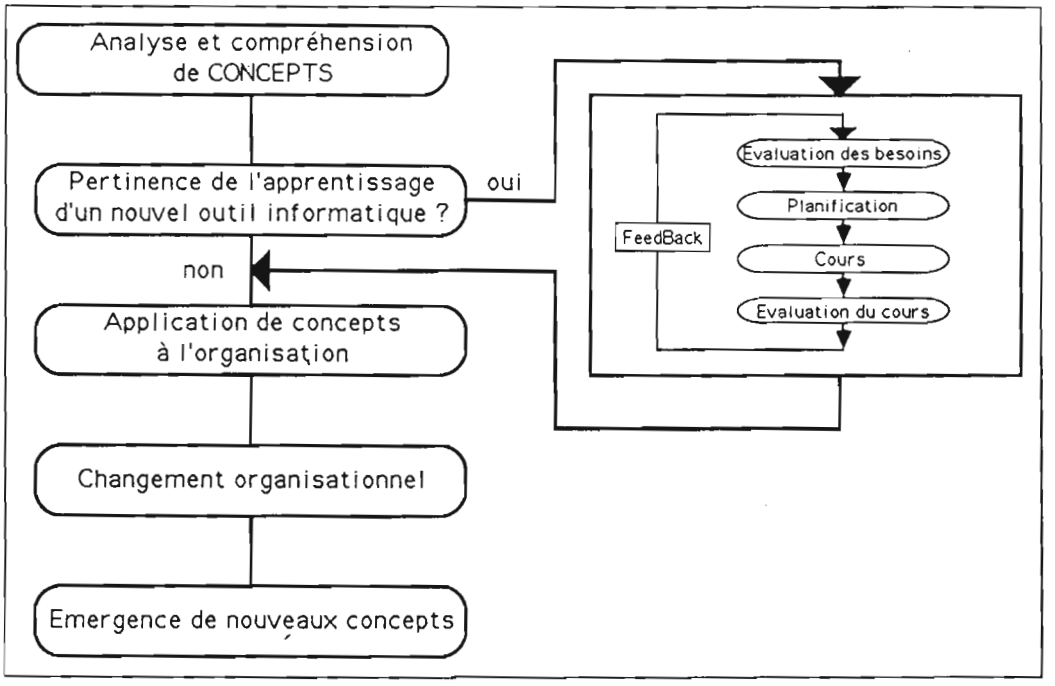
Quelques recommandations

Nous ne saurions clore cette réflexion sur la formation sans proposer au lecteur quelques points de repères. Ceux-ci constituent à notre sens des pierres angulaires qui risquent fort de devenir des pierres d'achoppement si l'on n'y prend pas garde. Les trois premières recommandations concernent la formation à l'informatique en général. Les trois suivantes sont plus axées sur une formation à l'apprentissage de concepts dans un champ donné.

La gestion des changements organisationnels

Dans notre introduction, nous avons dit que l'informatique permet l'informatisation de méthodes de travail. Elle peut être un vecteur de changement de ces méthodes de travail, car elle rend possible des opérations qui ne l'étaient pas avant l'informatisation et elle crée de nouvelles tâches.

La plupart des traitements de textes modernes permettent de faire des lettres personnalisées. Typiquement, il s'agit d'une lettre-type que l'on envoie à une série de personnes mais dont on veut personnaliser certains passages. Pour ce faire, le traitement de texte doit aller lire dans un fichier une liste d'adresse dans un certain format. Dans la lettre-type se trouve l'indication du nom de ce fichier d'adresses ainsi que certaines écritures conditionnelles ou des ordres de recopie de certains champs du fichier d'adresses. Il est probable que les secrétaires finiront par découvrir l'utilité de ce genre de commande et par l'employer de plus en plus fréquemment. Pour éviter la multiplication de fichiers



d'adresses, il peut être utile de créer une centrale d'adresses que gérera *un* responsable. Un programme central sur ordinateur central («mainframe») avec un codage des adresses peut s'avérer la solution idéale. Il faudra alors prévoir des connections entre les machines de traitement de texte et l'ordinateur gérant les adresses. Si celui-ci est un P.C., un réseau local fera probablement l'affaire. Cette solution donnera lieu à la mise en place d'un programme informatique, qu'il soit acheté sous forme d'un logiciel tout fait ou conçu par la fonction informatique. Une formation supplémentaire peut s'avérer nécessaire. La mise en place d'un tel système qui est en fait une base de données peut créer des problèmes organisationnels, de confidentialité par exemple. La encore une solution organisationnelle ou technique devra être mise en oeuvre. Et ainsi de suite.

Cette manière de procéder débouche sur une autre conception de la formation que nous décrivons dans le schéma ci-dessous.

La formation à l'informatique devrait pouvoir permettre l'accès à des *outils de changement organisationnels*.

Selon nous, la formation ne doit pas être nécessairement planifiée lors de l'apparition d'un nouveau logiciel. La solution évolutive nous semble l'apprentissage une bonne fois pour toute de *l'esprit* de la démarche informatique, solution qui s'oppose à la formation sans cesse recommencée à des modes opératoires changeants. Ce premier apprentissage doit aussi avoir comme objectif de faire comprendre, en prenant en considération autant que faire se peut l'histoire et le contexte du travail des utilisateurs, des concepts et des principes universels qu'utilisent depuis toujours les nouvelles technologies de l'information.

La mutation des images

Ce type de cours a aussi pour but de faire surgir une image plus réelle des informaticiens et de l'informatique. Il s'agit très clairement de créer une certaine *culture informatique*.

En soi, l'informatique n'est ni simple, ni compliquée, puisqu'elle est le fruit d'une démarche qui tend de plus en plus à s'affranchir de la technique. Des modes opératoires contraignants sont de moins en moins le fait d'impératifs techniques. Les utilisateurs doivent apprendre à devenir exigeants, mais cette exigence, pour être entendue, ne peut s'accompagner d'incompétence. Ils doivent être suffisamment décomplexés par rapport à l'informatique pour pouvoir refuser des programmes mal conçus.

Aujourd'hui, l'ergonomie informatique ne se limite certainement plus à l'intensité lumineuse des écran vidéo ou à la hauteur des claviers. Elle passe par une convivialité intelligente et respectueuse du travail de l'utilisateur. Par la compétence issue d'une formation conceptuelle, les utilisateurs devraient pouvoir intervenir comme acteurs dans les choix d'informatisation qui doivent être faits et suggérer des progrès techniquement réalisables. Cette attitude doit pouvoir déboucher sur un refus motivé de certaines applications non conviviales. La conscience du non déterminisme technique devrait normalement être une source d'un dialogue fructueux entre utilisateurs et informaticiens compétents.

L'explication en profondeur des erreurs

De nombreuses personnes baptisent leur machine d'un nom humain et lui associent une humeur dont la variation est fonction de critères plus ou moins fantaisistes.

Ainsi, un utilisateur peu compétent soutiendra mordicus que le fonctionnement correct de son imprimante à laser dépend de la température extérieure. En l'occurrence, elle dépend de réglages que la personne qui travaille avant lui sur la même machine effectue, personne qui vient travailler sur son imprimante uniquement par grande chaleur, quand il fait trop chaud dans son propre bureau.

Une autre remarque que, lors d'un tri d'un fichier d'adresses par code postal, une personne (toujours la même !) ne se trouve pas à sa place. Cela vient en fait de la confusion faite entre la lettre O et le chiffre 0 qui ont la faiblesse de se ressembler très fort. Lors d'un classement alphabétique par code, un programme logique classe 3O51 après 3052 puisque dans le code ASCII qui est l'alphabet de l'ordinateur, la lettre O se trouve après le chiffre zéro.

Afin de faire comprendre que le fonctionnement d'un programme n'est pas soumis à l'humeur des électrons qui circulent dans le processeur, il nous semble important de procéder comme suit. Tout d'abord faire identifier l'erreur d'une manière certaine par quelqu'un de compétent; une erreur est identifiée quand elle est reproductible à souhait. Ensuite, il s'agit de montrer à l'utilisateur que la cause qu'il croit avoir identifiée n'a en fait aucune influence sur l'occurrence de l'erreur. Enfin il suffit d'expliquer la source véritable de l'erreur et de montrer, preuve à l'appui, que l'action sur cette cause supprime cette erreur.

Versus la boulimie logicielle : le bon outil correspond au besoin.

Afin de parvenir à vendre la plus grosse machine possible, les commerciaux ont su, spécialement en informatique, jouer sur l'angoisse du trop petit. Ce qui est vrai pour le matériel l'est peut-être plus encore pour le logiciel.

Des clubs informatiques permettent à leurs membres d'acquérir par échange (et naturellement sans licence) des centaines de programmes informatiques, qu'il s'agisse de logiciels «déplombés[19]» ou copiés, de programmes du domaine public acquis en «shareware[20]» ou en «freeware[21]». Comme si la culture informatique se mesurait à l'aune du nombre de programmes détenus...

La bonne culture informatique est la connaissance des différents types de programmes informatiques (tableurs, grapheurs, programmes de mise en page, systèmes experts, gestionnaires de bases de données, programmes de communication par modem, etc...). Afin de parvenir à effectuer un réel choix parmi ces différents outils, l'utilisateur doit posséder une bonne perception des différentes fonctionnalités qu'il est en droit d'attendre de telle ou telle famille de programme.

Ceci permet, par exemple, pour rédiger une facture, de ne pas se servir d'un traitement de texte mais bien d'un tableur, plus adéquat pour ce type de travaux.

Résister à la tentation de l'opérationnalité immédiate

Lorsqu'un utilisateur vient en dépannage logiciel chez un délégué de la fonction informatique, il revendique bien souvent la solution ponctuelle d'un problème urgent[22]. Tout comme le malade ne demande pas vraiment à son médecin la guérison de sa maladie mais plutôt la disparition des symptômes qui le gênent. Pourtant cette rencontre peut être l'occasion idéale de prévenir d'autres erreurs du même type.

Pour l'informaticien surchargé, cela peut paraître un bon calcul de trouver la ficelle technique mystérieuse et rapide qui permet à l'utilisateur de résoudre son problème ponctuel... et de refaire des erreurs du même type pour lesquelles il ira de nouveau déranger un délégué de la fonction informatique.

Prenons un exemple typique. Un utilisateur vient demander quelle commande exécuter pour se positionner à la 197ème ligne d'un fichier. L'informaticien lui griffonne sur un bout de papier la commande suivante : ESC < CTRL/U 1 9 6 CTRL/N[23]. L'utilisateur teste cette formule magique sur son clavier et abracadabra, tout se déroule comme l'informaticien l'avait prévu. S'il n'est pas né de la dernière pluie, il aura vite deviné la commande permettant d'aller à la 223ème ligne. Quelques jours plus tard, il risque fort de demander la commande pour aller 200 lignes plus haut que la position du curseur.

Si l'informaticien lui avait expliqué la commande, il pourrait le faire sans peine. En effet, il suffit de savoir que la commande CTRL/U suivie d'un nombre et d'une commande permet de répéter automatiquement cette commande ce nombre de fois. La commande ESC < permet de se positionner au début du fichier et CTRL/N permet de descendre d'une ligne. Autrement dit, pour aller à la 197ème ligne d'un texte il suffit d'aller au début du texte et d'aller 196 fois à la ligne suivante. L'utilisateur qui a compris cela aura vite fait de taper CTRL/U 2 0 0 CTRL/P pour remonter le curseur de deux cents lignes...

Préférer l'intégration historique au jargonnage technique

Il nous paraît essentiel en clôturant cet article d'insister une dernière fois sur l'importance de l'histoire dans tout processus d'apprentissage. Nous serions tenté de dire qu'il faut substituer aux métaphores des *isophores*. C'est dire qu'il faut présenter l'informatique comme un processus de communication que l'utilisateur connaît déjà plutôt que comme une technique que rien dans son métier ne lui permet de palper et de mesurer.

Imaginons que nous devions expliquer à un papou tiré du plus profond de sa forêt vierge ce qu'est l'aviation. La bonne démarche serait-elle de lui apprendre d'abord les mathématiques et de lui donner des cours de résistance de matériau, d'aérodynamique, etc. ? Comprendrait-il mieux si on lui mettait directement un manche à balai entre les mains en tentant, tant bien que mal, de lui expliquer l'horizon artificiel, l'usage du palonnier et l'utilité du siège éjectable ? Nous avons expliqué dans notre introduction que l'apprentissage est un processus de *compréhension* qui se raccroche à l'histoire de l'élève. Dès lors, le plus simple ne serait-il pas de repartir des oiseaux que notre papou connait déjà? L'approche que nous préconisons ici se propose de partir, non pas de l'informatique mais de l'utilisateur et donc des images qu'il peut en avoir du fait de sa vie en Société.

L'idéal nous semble être que le cours puisse être mis sur pied par des agents dont la tâche d'enseignement n'est qu'une partie mineure de leurs activités. Nous recommandons durant ce cours d'appliquer les différentes fonctionnalités que nous avons détaillées ci-dessus à un exemple tiré, non pas d'un manuel étranger, mais d'un problème usuel dans le travail des utilisateurs. L'idéal serait même que les élèves puissent, après quelques heures de cours, apporter eux-même un problème courant qu'ils souhaitent résoudre.

Nous incitons donc les formateurs qui liront ces lignes à se défier tout particulièrement des exemples jargonneux qui pullulent dans les manuels soit-disant d'apprentissage de certains logiciels issus d'outre atlantique. L'exemple servant de référence durant l'apprentissage doit être un problème concret et fréquemment rencontré par les utilisateurs.

JEAN-MARC DINANT
CENTRE DE RECHERCHE INFORMATIQUE ET DROIT
FUNDP-NAMUR

[1] A. VAN LAETHEM, citée dans Data Décision Nov 1991, p.31.

[2] F.PAVÉ, *L'illusion informaticienne*, Editions l'Harmattan, Paris, 1989, p.11.

[3] F. PAVÉ cite simplement les qualités de bêtise et de discipline, opposant ainsi subrepticement la machine à l'être humain dont on suppose l'intelligence et le sens de l'initiative. La rapidité sans cesse croissante du traitement informatique est bien évidement *la* qualité qui a permis à l'ordinateur de supplanter l'être humain dans de nombreuses tâches matérielles. On peut aussi contester l'anthropomorphisme de cette métaphore en se demandant si on peut dire d'une ampoule qu'elle est bête et disciplinée lorsqu'elle s'allume suite à la pression d'un interrupteur. Cela revient quelque part à se poser la question du sexe des anges ou de la couleur de l'informatique. Nous serions plutôt tentés de dire que l'ordinateur sort du champ de l'intelligence (fût-elle artificielle) et de l'obéissance. Ces notions recouvrent en fait une caractéristique à la fois intéressante et gênante : celle du déterminisme.

[4] Voir à ce sujet : Ph. BRETON, *La tribu informatique*, Editions Métaillé, Paris, 1990, pp. 144-151.

[5] Le silicium (composante principale du sable) est le matériau de base des processeurs et mémoires.

[6] Imagination que PASCAL décrit déjà en 1658 comme étant la «partie dominante de l'homme, cette maîtresse d'erreur et de fausseté, et d'autant plus fourbe qu'elle ne l'est pas toujours, car elle serait règle infaillible de vérité, si elle l'était infaillible du mensonge» (Pensée n° 44 (82))

[7] Que le lecteur ne croie pas trop vite que nous attribuons aux utilisateurs d'informatique une mégalomanie déplacée. Le Père DUBARLE, dans un article publié dans le journal *Le Monde* en 1958 parlait déjà de l'ordinateur comme *d'une machine à gouverner, du premier grand relais du cerveau humain, d'un évident surclassement des pouvoirs organiques du cerveau de l'homme...* (cité par Ph. BRETON in *La tribu informatique*, Editions Métaillé, Paris, 1990).

[8] Ce *parfois* mériterait qu'on s'y attarde. On se rend compte *a posteriori* que nombre d'activités humaines ne peuvent réussir que grâce à l'exercice du bon sens. Etrangement, ce bon sens est tellement imbriqué en nous qu'il ne nous semble pas *a priori* indispensable à la réussite. S'il l'était systématiquement, il ne s'agirait plus évidement de bon sens. Et c'est le bon sens lui-même qui nous permet de comprendre ce qu'est le bon sens et pourquoi il s'avère nécessaire dans un cas précis; tout comme il nous permet d'utiliser un dictionnaire qui définit une définition. Sans ce bon sens le dictionnaire n'est qu'une immense tautologie inexploitable puisqu'*avant* de commencer à l'utiliser, nous aurions du comprendre (sans le dictionnaire) ce qu'est une définition.

[9] Le compilateur est un programme. Il doit être écrit lui-même dans un certain langage, différent de celui qu'il est censé compiler. Il n'en est rien. Un compilateur a cette caractéristique étonnante de pouvoir se compiler lui même. On appelle *code source* ou *source* le programme écrit dans le langage de programmation. On appelle *code objet* ou *exécutable* le résultat produit par le compilateur et qui est en code binaire, directement exécutable par le processeur.

[10] Cette exemple nous permet facilement de mieux comprendre ce qu'est un bogue. Dans ce cas, un programmeur distrait une fraction de seconde aurait pu écrire SB = ST + PRIME

[11] DOUGLAS HOFSTADER, GÖDEL, *Escher et Bach ou les brins d'une guirlande éternelle*, InterEditions, Paris, 1985, pp. 57-62 : les isomorphismes induisent la signification.

[12] L'ordinateur a été électriquement cablé d'une manière immuable pour exécuter, lors de sa mise en service, la première instruction d'un petit programme contenu en mémoire morte (le BIOS : Basic Input Output System) qui contient des instructions qui vont permettre de charger en mémoire vive (RAM) le système d'exploitation contenu sur disque dur ou disquette. Dès ce moment et jusqu'à l'arrêt de la machine c'est ce programme du système d'exploitation (O.S. = Operating System) qui va gérer l'exécution des différents programmes.

[13] F. PAVÉ, Ibidem

[14] R. JACOBSON, «*Essais de linguistique générale*», Editions de Minuit, Paris, 1963.

[15] «*Les logiciels de traitement de texte permettent de supprimer facilement une lettre, un mot, un paragraphe et de les faire réapparaître ici et là, la mise en page se réorganisant automatiquement. Les ajouts, corrections, modifications ne nécessitent plus la réécriture d'une page, voire d'un texte entier, ou le laborieux maniement de la colle et des ciseaux....*» in P. LEVY, *La machine univers*, Editions La Découverte, Paris, 1987.

[16] Ces disques sont appelé WROM : Write Once, Read Many.

[17] D'où son nom : «return».

[18] Ce concept de compétence linguistique constitue cette capacité que possède un individu de comprendre un message ou un énoncé qu'il n'a jamais rencontré auparavant. De même en informatique, un adulte ayant compris le langage de l'informatique devrait être à même de comprendre seul les fonctionnalités d'un programme qu'il n'a jamais rencontré.

[19] illégal : il s'agit d'enlever d'un programme les instructions permettant à celui-ci de ne pas se laisser copier d'un support informatique sur un autre. Après déplombage, les programmes ainsi déprotégés se copient sans problème d'un support sur un autre.

[20] Tout utilisateur appréciant un programme communiqué librement et l'utilisant régulièrement est prié d'envoyer une petite somme à son auteur. Ce système marche bien surtout aux USA.

[21] Le logiciel est gratuit et distribuable à tous.

[22] Voir à ce propos l'article *rencontre avec un gestionnaire de formation*, dans ce numéro du JRI.

[23] Cet exemple est authentique et concerne l'éditeur de texte «Emacs».